Defining Security Monitoring SLAs in IaaS Clouds: the Example of a Network IDS

Amir Teshome Wonjiga, Louis Rilling, Christine Morin

Defining Security Monitoring SLAs in IaaS Clouds: the Example of a Network IDS

Amir Teshome, Louis Rilling, Christine Morin

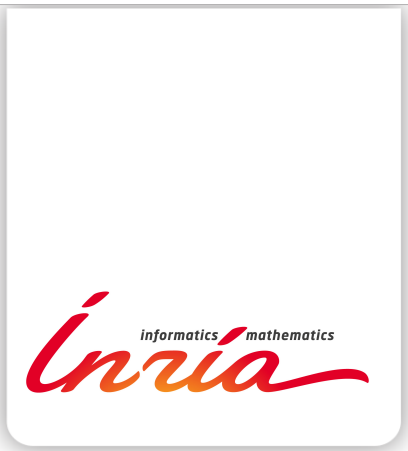
RESEARCH REPORT

N° 9263

March 2019

Project-Team Myriads

ISSN 0249-6399 ISRN INRIA/RR--9263--FR+ENG



Defining Security Monitoring SLAs in IaaS Clouds: the Example of a Network IDS

Amir Teshome*, Louis Rilling†, Christine Morin*

Project-Team Myriads

Research Report n° 9263 — March 2019 — 37 pages

Abstract: In an IaaS cloud the physical infrastructure is controlled by service providers, including its security monitoring aspect. Clients hosting their information system need to trust and rely on what the providers claim. At the same time providers try to give assurance for some aspects of the infrastructure (e.g. availability) through service level agreements (SLAs). We aim at extending SLAs to include security monitoring terms. In our previous study [1] we proposed a verification method for security monitoring SLAs describing the performance on an network intrusion detection system (NIDS). In this paper we address the problem of security monitoring SLA definition, specifically for the case of NIDSs in cloud. We present the following contributions. First we propose a security monitoring service description with relevant key performance indicators (KPIs). Second we propose an extension to an SLA language called *CSLA* [2], in order to have a standard method to define security monitoring SLAs. Third the KPIs used to describe performance of NIDS take a *base rate* parameter, representing the rate of attacks in the monitored network traffic. However, the value of the base rate is unknown at the time of SLA definition. In order to address this contradiction, we propose a model building method and the model is used in the SLA definition. The model is used to estimate the expected performance depending on the base rate. Fourth, since there is a large number of vulnerabilities among all software products possibly used by tenants, defining an SLA requires lots of performance evaluation tests, which makes the process impractical. To address this we propose a method based on rules clustering which builds a knowledge base for NIDS performance for a large number of vulnerabilities. Finally, we present experiments showing the feasibility of our methods on performance estimation and clustering of NIDS rules. We also present analysis on the shortcomings of the proposed method.

Key-words: Clouds, security monitoring, SLA, SLA language, vulnerabilities, NIDS, NIDS performance, rule interference

* Univ Rennes, Inria, CNRS, IRISA
† DGA-MI

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Définition de SLAs pour la supervision de la sécurité dans les clouds de type IaaS : exemple d'un IDS réseau

Résumé : Dans un cloud de type IaaS, l'infrastructure physique est contrôlée par les fournisseurs de services, y compris sur l'aspect surpervision de la sécurité. Les clients hébergeant leur système d'information doivent se fier à ce que les fournisseurs affirment. Dans le même temps, les fournisseurs essaient de donner une assurance sur certains aspects de l'infrastructure (par exemple la disponibilité) par le biais de contrats de niveau de service (*Service-Level Agreement* ou SLA). Notre objectif est d'étendre les contrats de niveau de service afin d'y inclure des aspects de supervision de la sécurité. Dans notre étude précédente [1], nous avons proposé une méthode de vérification du respect d'objectifs de supervision de la sécurité dans les SLAs, ces objectifs décrivant la performance d'un système de détection d'intrusion dans le réseau (NIDS). Dans le présent document, nous abordons le problème de la définition des SLAs portant sur la supervision de la sécurité, en particulier dans le cas des NIDS dans les clouds. Nous présentons les contributions suivantes. Tout d'abord, nous proposons une description du service de supervision de la sécurité avec des indicateurs clés de performance (*Key Performance Indicators* ou KPIs) pertinents. Deuxièmement, nous proposons une extension d'un langage de SLA appelé *CSLA* [2], afin d'avoir une méthode standard pour définir les SLA de supervision de sécurité. Troisièmement, les KPIs utilisés pour décrire la performance des NIDS prennent en paramètre le *taux d'attaques* dans le trafic réseau surveillé. Toutefois, la valeur du taux d'attaques est inconnue au moment de la définition d'un SLA. Afin de résoudre cette contradiction, nous proposons une méthode de construction d'un modèle et le modèle est utilisé dans la définition du SLA. Le modèle permet d'estimer la performance attendue en fonction du taux d'attaques. Quatrièmement, comme il existe un grand nombre de vulnérabilités parmi tous les produits logiciels éventuellement utilisés par les utilisateurs du cloud, la définition d'un SLA nécessite de nombreux tests d'évaluation des performances, ce qui rend le processus difficilement applicable. Pour remédier à cela, nous proposons une méthode fondée sur le regroupement de règles qui permet de construire une base de connaissances sur la performance des NIDS pour un grand nombre de vulnérabilités. Enfin, nous présentons des expériences démontrant la faisabilité de nos méthodes d'estimation des performances et de regroupement des règles de NIDS. Nous présentons également une analyse des limitations de la méthode proposée.

Mots-clés : clouds, supervision de la sécurité, SLA, language de SLA, vulnérabilités, NIDS, performance de NIDS, interférences entre règles de NIDS

1 Introduction

Before the introduction of cloud computing, organizations used to host their own computing resources (networks, servers, storage, applications, and services). Using clouds, organizations (called tenants) benefit from cost reduction (in both building and management) but they also face new problems in terms of security. When moving to a cloud tenants lose full control of the information system infrastructure and must trust the service provider. The provider is in charge of monitoring the physical infrastructure and providing the required service to tenants. Clients hosting their information system need to trust and rely on what the providers claim. At the same time providers try to give assurance for some aspects of the infrastructure (e.g. availability) through service level agreements (SLAs). However, as of today, service providers do not give assurance on the security monitoring of the hosted information systems. Our goal is to extend current cloud SLAs to include security monitoring terms.

Security monitoring is the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions [3]. The goal of collecting and analyzing events and generating indicators is to *detect* and prevent intrusions. In addition, when prevention eventually fails, the goal is to *respond* to incidents as quickly as possible and understand how the intruder achieved its attack and what damage it made. Different types of security monitoring devices and techniques are used for different components.

In a previous study [1] we separated the SLA lifecycle in three phases namely *SLA definition and negotiation, SLA enforcement and SLA verification*. We proposed an SLA verification method for NIDSs in the cloud. In this paper, we describe in detail how we can achieve the SLA definition and negotiation phase.

SLA definition is a pre-negotiation phase where service providers draft SLA templates according to the services they provide. SLA definition happens before any contact with a tenant, providers assess their performance and prepare templates to offer for potential tenants. The negotiation process starts after a tenant discovers a provider and shows an interest in the provided service. Activities in the SLA definition phase include describing and preparing quantifiable objectives for the provided service. In this phase the service description should clearly address the type of service(s) covered under that SLA and providers precisely measure their performance and generate a *Key Performance Indicator* (KPI).

From the provider perspective, SLA definition is an advertisement for the provided service. It should be clear, precise and it should attract targeted customers. Service providers publish and advertise their services using the draft template SLAs. Tenants are also expected to perform some activities before signing off an agreement with a service provider. In [4] the Cloud Standard Customer Council (CSCC) describes ten steps that help to compare multiple cloud providers or to negotiate terms with a selected provider.

SLAs can address the different cloud service models. In this paper we assume an IaaS system. In our previous study we described the lack of security monitoring terms in SLAs. The majority of existing SLAs addresses system availability but fails to give a guarantee on other aspects of security. We approach this problem by including security monitoring terms into SLAs. Security monitoring is a process and different tools are used to perform this process. In our work *Network Intrusion Detection Systems* (NIDSs) are used as a monitoring device and security monitoring SLAs are defined for NIDSs.

In our context, security monitoring SLA terms refer to SLA terms that are designed to give guarantees on the performance of NIDSs. Our work follows a user-centric approach in the sense that it gives more choices and flexibility for tenants. Tenants can specify which vulnerabilities to be monitored, can verify whether a specified objective is reached and can take actions if there is an occurrence of SLA violation.

In the next section, we describe the objectives and problems that are addressed in this paper. Specifically, how can we formally describe security monitoring SLA terms? What are the relevant metrics to describe the performance of an NIDS and how to include these metrics in security monitoring SLAs? In the presence of tens of thousands of vulnerabilities how a provider can prepare tenant-tailored SLA. To this end, we show how to determine the performance of an NIDS on a subset from tens of thousands of vulnerabilities and the effect on the performance of an NIDS regarding an increased number of vulnerabilities. Additionally, we present our approach to address the issue of defining security monitoring SLA terms with appropriate NIDS metrics. In this paper, we also provide experimental results to validate the proposed ideas.

2 Objectives and Problem Description

In this section, we present the objectives and we describe problems that we address. SLAs are fundamental components of the cloud computing model. As a result, clearly defined SLAs facilitates the regulation process throughout the service life-cycle. In this paper, we aim to achieve the following objectives.

2.1 Objectives

In general, the objective of this paper is to provide a mechanism which enables to define security monitoring SLA terms. More fine grained objectives of this paper include:

- The mechanism should follow a user-centric approach, i.e. users should be able to participate in the process, have choices and flexibility on what type of vulnerabilities should be addressed.
- For a tenant to describe her/his needs, the mechanism should require fairly little knowledge about the details of the system which is outside their own environment, i.e it should not require technical knowledge about the system controlled by the provider.
- From providers the perspective, the mechanism should enable them to prepare custom tailored SLA for each tenant according to their requirements.
- The mechanism should allow describing the performance of monitoring devices with fine-grained, relevant metrics. It should also allow describing the process of computing composite metrics.

In order to achieve these objectives, we need to tackle some problems. We have identified the following problems.

2.2 Problem Description

The cloud comes with a characteristic of pay per use which magnifies the significant variation in consumer needs. Hence, SLAs have to be created for each tenant by a negotiation process. The communication between the provider and tenants should be in a standard language. This provides assistance for tenants to communicate and compare SLA offers from different service providers without the need to adapt another communication standards or language.

SLA definition should incorporate components like *Service Description, Service Level Objectives (SLO), Parties, Penalties*. For security monitoring terms the service description should be able to express *vulnerabilities* as they represent the finest granular concept in our SLA description. In addition, a vulnerability should be related to other services like products (software)

where the vulnerability exists and infrastructure where the product is running. Currently, to the best of our knowledge, there is no SLA language that can satisfy these requirements.

The KPI describing the objectives of a monitoring device should be relevant for the specific device used, in our case for NIDSs. For our use case two characteristics of a metric are set as prerequisites before being used as SLO for terms describing NIDSs performance. These are (i) the metric should take *base rates* into consideration. Axelsson [5] showed the base rate fallacy problem for NIDSs and indicated the importance of base rate. (ii) it should be a single valued metric i.e not a combination of two or more metrics. Comparing coupled metric values from different providers requires analysis and finding trade-offs. Hence, from tenants perspective, it is easier to have a single unified metric. The non-deterministic nature of the base rate makes it challenging to define security monitoring SLAs. An SLA should not be defined for specific values of the base rate because, most of the time, every occurrence of an attack has a different rate, i.e. different base rate.

Many languages have been developed to define SLAs for different types of services, e.g. WSLA [6], WS-Agreement [7], SLAng [8], SLAC [9], CSLA [2]. Some of these languages were developed for web services before the cloud (e.g. WSLA and WS-Agreement) or are DSLs specifically designed for clouds (e.g. SLAC and CSLA). However, none of these languages can describe security monitoring terms with the requirements discussed above.

From the providers perspective, there are two facts which make the SLA definition process challenging and needs consolidation. The first one is the fact that tenants need custom tailored security monitoring services. This is directly related to the nature of cloud services. By design the cloud is flexible, and tenants can configure it as they want. As a result, no one security monitoring configuration can fit all tenants needs.

The second challenge is that there are thousands of vulnerabilities and almost all tenants are interested in a few of these vulnerabilities. One tenant can be interested in three vulnerabilities and another tenant can be concerned with twenty vulnerabilities. This is intuitive considering tenants are running different services on various types of configuration. Service providers should know the performance of their monitoring device on every subset of vulnerabilities. But performing a direct measurement for every subset of vulnerabilities is not practical because it requires a large number of computations.

Moreover, in practice the number of vulnerabilities may vary from time to time. For example, a patch is available for a given vulnerability and monitoring that vulnerability will no longer be necessary. Alternatively, the other way round, a new vulnerability concerning a tenant may be discovered and the tenant requests monitoring against this vulnerability. In our work, we do not address a dynamic change in the SLA, Section 7.4 describes how to deal with such changes.

To perform the monitoring task using NIDSs, rules should be crafted and added to the configuration of an NIDS. The number of rules configured in an NIDS affects its performance, i.e as the number of rules increases the performance decreases. To make an NIDS custom tailored means to configure a tenant's NIDS with rules only concerning that tenant. Hence, service providers should know the performance of their monitoring device for every possible combination of available vulnerabilities. Given the high number of vulnerabilities it is not practical to do the evaluation task for every combination.

2.3 Contributions

In summary in this paper we present the following contributions:

- We propose a security monitoring service description with relevant KPIs, specifically an SLO for NIDSs with a single unified metric which takes the base rate into account is used

to describe the performance of an NIDS. Section 3 describes the KPI used in our SLA and other components of security monitoring SLAs.

- To address the lack of a standard language for security monitoring SLAs, specifically to be able to describe vulnerabilities and their relation with other services, we propose an extension to *CSLA* [2], a DSL used to describe cloud SLAs. Section 4 describes the extension in detail.
- Defining a KPI which relies on the base rate is challenging. This is because measuring the value of the base rate before the occurrence of an attack is difficult, if not impossible. We propose an interpolation-based mechanism which takes performance points from known base rates as an input, build a model and estimate the expected performance using that model for new base rate values. Section 5 presents the method in detail.
- In order to prepare SLO templates, we propose a method which builds a knowledge base for NIDS performance for a large number of vulnerabilities. The method aims to reduce the number of required performance evaluations since evaluating the monitoring device over all possible combination of vulnerabilities is not practical. The method is described in Section 6.

The next section presents the components of security monitoring SLAs.

3 Components of Security Monitoring SLAs

Like any other type of cloud SLAs, security monitoring SLAs contain the components like *Service Description, Service Level Objectives (SLO), Parties, Penalties*. Besides the anatomy similarity with other clouds SLAs the content of security monitoring SLAs differs from others in few ways. In this section we present components of security monitoring SLAs, classes of SLO that we consider and the KPI used to describe the performance of NIDSs. Security monitoring SLA components include:

- *Service Description:* The security monitoring service is used to monitor existing vulnerabilities in a product. Hence, the service definition correlates three components (i) product: a software running in tenants infrastructure that contains vulnerability (ii) vulnerabilities: as a product may contain more than one vulnerability it should be specified which vulnerabilities are covered in the agreement. (iii) infrastructure: where the product is running. NIDSs require information about the infrastructure where they perform the monitoring task. Specifically, an NIDS needs IP addresses of machines to be monitored.
- *Parties:* are participants in the agreement. Primarily, it includes providers and tenants. Supporting parties (e.g. auditor) can be included depending on the need.
- *Penalties:* apply on violating party in cases of SLO violation. In our SLA definition a penalty can be defined in two forms. (i) Fixed type penalty which applies a fixed amount of retribution per occurrence of violation (ii) Function type where the penalty is based on a predefined function.
- *Service Level Objectives (SLO):* describe the guaranteed level of performance. For an NIDS, it shows how much a given NIDS is effective in detecting the vulnerabilities listed in the service description. The performance of an NIDS is described using C_{ID}. The next subsections provide details on why the C_{ID} is used and class of SLOs that are considered in our work.

Application	**Version**	**Attacks**
Apache	Apache/2.4.7 (Ubuntu)	Denial of service (DoS) and port scan
Mysql	14.14 Distrib 5.6.31	Brute force access
WordPress(WP)	V. 4.4.5	None
Instalinker (WP plugin)	V. 1.1.1	Cross site scripting (XSS)
Custom Contact Forms	V. 5.1.0.2	SQL injection

Table 1: Example of service list and attack types figuring in an SLO

3.1 KPI for Security Monitoring SLO

As described in Section 2.2, for a metric to be used in our SLO two features are required, namely a single unifying metric and the base rate. Intrusion Detection Capability (C_{ID}) satisfies both of these features (see [1]). The C_{ID} measures the ratio of the mutual information shared between the inputs and outputs of an NIDS and the entropy of the input. Mutual information measures the uncertainty in the input (i.e whether a given input packet is part of an attack or not) after knowing the NIDS output. Normalizing this value with the initial uncertainty of the input (i.e entropy) produces C_{ID}.

The definition of SLA describes how to compute the expected C_{ID} value from basic metrics (TPR and FPR) and a base rate. Since it is not possible to know the values of base rate in advance, SLA definition does not contain exact values of base rate. Section 5 presents how providers can offer SLO based on unknown base rate values.

3.2 Class of SLOs Considered

In the SLA definition phase providers prepare SLO templates. These templates describe the expected effectiveness of NIDSs using the C_{ID} metric while they are configured to monitor vulnerabilities. Providers prepare templates taking only *known* vulnerabilities into account. Tenants may require to be monitored against new or emerging vulnerabilities, but from the providers perspective, it is risky to promise the performance of monitoring devices for unknown vulnerabilities. Section 7.4 describes the issues related to unknown vulnerabilities.

Tenants describe their needs by listing vulnerabilities of their interest. The final SLA contains the list of vulnerabilities that are at the intersection between tenants requirements and security monitoring services proposed by the provider. The agreement describes the list of services to be monitored, the list of known vulnerabilities, and expected performance. Services are described with an application(s) that are used to provide that service. In addition, vulnerabilities are related to the applications version. Once the agreement is signed tenants' infrastructures are monitored against attacks which can exploit the listed vulnerabilities.

Example of services considered to be included in an SLA are described in Table 1. The table shows a list of vulnerabilities and attacks to be monitored including a Denial of Service (DoS) and a port scan against the Apache web server, a brute force access to the Mysql database server, a cross-site scripting (XSS) attack against Instalinker WordPress plug-in and an SQL injection against Custom Contact Forms (also a WordPress plug-in). An example SLA including this service is presented in the next section.

The next section presents a formal language to describe cloud security monitoring SLAs.

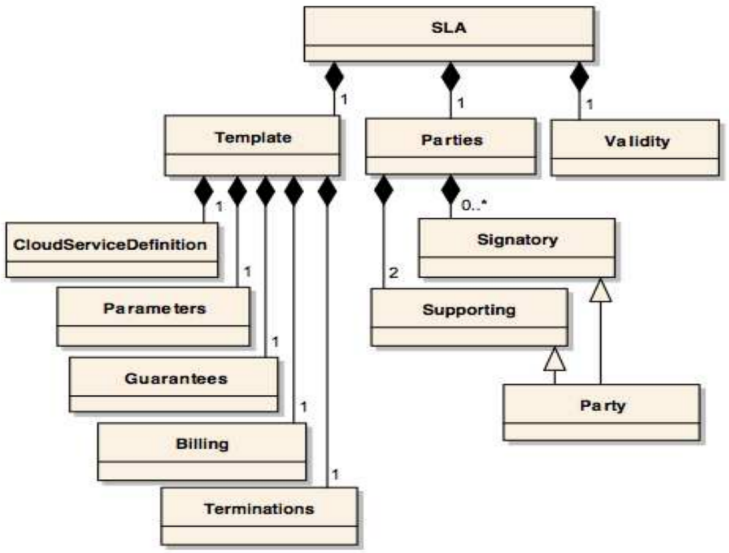


Figure 1: CSLA meta model [2]

4 Extended Cloud Service Level Agreement (ECSLA)

As presented in Section 2.2, one of the problems for realizing security monitoring SLAs is the lack of formal language to define an SLA. In practice there are few SLA languages, some of these languages were designed before the cloud era, and they do not address the distinctive characteristics of clouds. Languages designed specifically for clouds fail to include functionalities to describe security features, with the exception of availability. Most of the languages e.g. SLAC [9] and CSLA [2] target functionality of a service like the response time.

To define security monitoring SLAs we need a language capable of describing vulnerabilities to be monitored and the relationship between vulnerabilities and other services, i.e. the relationship between a vulnerability and a software, showing on which software the vulnerability is residing, and relationship between a vulnerability and infrastructure, showing on which infrastructure the vulnerable software is running. To achieve this we propose an extension to a cloud Domain Specific Language (DSL), CSLA [2]. Before describing the proposed extension, we first present the CSLA language and the reason why we selected CSLA. An example SLA is presented at the end of this section.

4.1 CSLA

CSLA [2] is a DSL designed to address the needs of cloud SLA. Specifically, CSLA tries to take into account the dynamic nature of clouds by introducing SLA properties to tolerate violation. It also adopts a dynamic penalty model described in [10]. These properties allow service providers to tolerate fluctuation in SLO.

Figure 1 shows the meta-model of CSLA. SLA in CSLA contains three sections. These are *parties, validity* and *template*. *Validity* describes how long the agreement is valid and *parties* describe who is bound by the agreement. In CSLA there are two types of parties *Signatory parties* (service provider and service customer) and *Supporting parties* (e.g., trusted third party). *Templates* are structural models for SLA. A template contains five sections namely Services

definition, Parameters, Guarantees, Billing and Terminations.

- Services definition: describe services following the standard cloud service models (SaaS, PaaS or IaaS). It uses the Open Cloud Computing Interface (OCCI) standard [11] for IaaS services definition.
- Parameters: used to define variables that are relevant to describe *Metric*, *Monitoring* and *Schedule* elements in other sections of the agreement.
- Guarantees: are elements containing the expected objectives. It contains four elements *Scope, Requirements, Terms and Penalties.*
  - *Scope*: specify services from the agreement that are covered by this guarantee.
  - *Requirements*: conditions that are expected to be fulfilled in order to achieve the objectives.
  - *Terms*: contain a set of guarantee terms connected by *'And'* or *'Or'* terms. Each term contains objective(s) that are defined by *expression* and *precondition.* A *priority* is also defined for each objective, and users have the option to set their preferences. An *expression* is characterized by different properties including a *metric*, a *comparator* and, a *threshold.* *Monitoring* and *Schedules* are also defined to specify how and when to evaluate the metrics. *Fuzziness* and *confidence* values are defined under the *expression.* As described in [2], these are parameters describing service degradation to deal with unpredictable environments.
  - *Penalties*: are applied in cases of SLA violation. *Constant* and *function* based penalties can be applied.
- Billing: describes the billing method for the provided service.
- Terminations: describes the termination procedure. These procedures will be followed depending on the *validity* section, specifically according to the date on *effectiveUntil* parameter of *validity* section.

The full syntax of CSLA is presented in [12].

4.2 ECSLA

The service description section of CSLA describes SaaS, PaaS or IaaS services. This is not enough to describe security monitoring services. We propose a generic extension to the service description section. Our extension helps to enable definitions for other types of services. In our case, the generic service definition is used to define the *SecurityMonitoring* service.

Figure 2 shows the model diagram of ECSLA. The dotted rectangular box shows the part from original CSLA. The *MacroService* definition represents a generic service class which can be extended to any type of class. The concept of using a generic class as part of the SLA structure is used in other languages like in SLA* [13]. The *SecurityMonitoring* service contains three sections namely *product, vulnerability,* and *infrastructure.* The *product* and *infrastructure* are defined in the original CSLA while the *vulnerability* is a new feature in ECSLA. A *product* is a general class to described software and platforms. Some properties of a product include *name, version, distribution, price* etc. Two types of cloud infrastructures are defined namely *Compute and Storage.* A *vulnerability* is characterized by *Id, Common Vulnerabilities and Exposures (CVE)* and *description.* The CVE is a list of vulnerability database entries, each containing

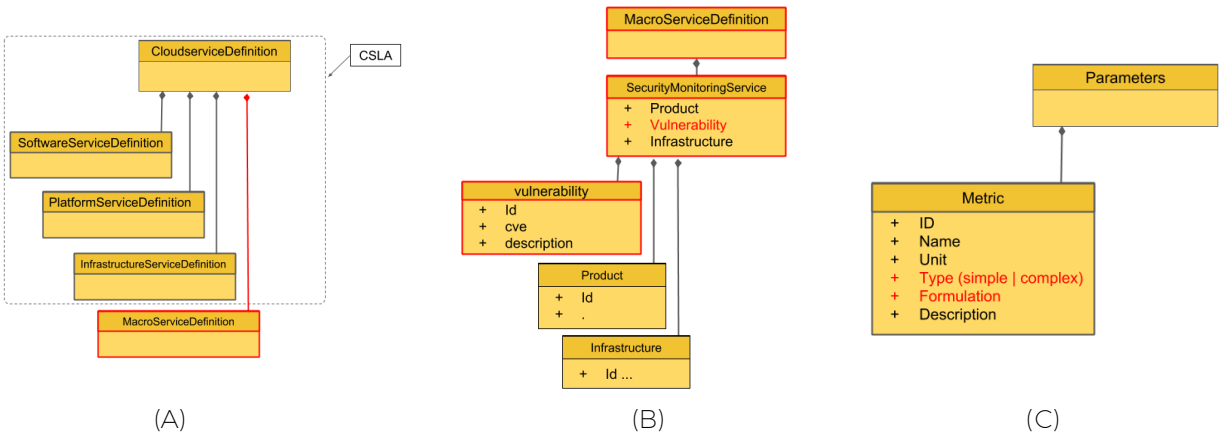


Figure 2: ECSLA model diagram (A) Macro service extension, (B) Security monitoring services and (C) Metrics

an identification number, a description, and at least one public reference for publicly known cybersecurity vulnerabilities. CVE entries are used in numerous cybersecurity products and services from around the world [14]. We use the *CVE ID* to characterize vulnerabilities in ECSLA. However, some vulnerabilities may have no matching CVE, i.e when the vulnerability is not added to the database list. In that case, in ECSLA the vulnerability is characterized by *Id* and *description*.

In addition, in the original CSLA, the *parameter* is used to define variables that are required to describe metrics. The *metric* is described with attributes *id, name, unit,* and *description*. In the original CSLA, if a complex metric (computed from basic metrics) is used to describe an SLO, there is no way to define its formulation. Describing the process to compute a complex metrics is important in order to clarify misunderstandings. Thus, we extended the metrics definition of CSLA in order to include such a feature. Figure 2(C) shows the proposed extension in metrics definition. The *type* and *formulation* attributes are added. The *type* can be either *simple* or *complex*. The *formulation* formally describes how a *complex* type metric is computed. If a metric is of a *complex* type, then *formulation* shows how to compute that metric using basic metrics. The *formulation* can contain the actual formal definition or a reference to a formal definition. Appendix 9 shows the formal XML model for ECSLA.

4.3 Example Security Monitoring SLA

In this section, we provide an example of a security monitoring SLA showing the concepts described above. We consider the services described in Section 3.2 (see Table 1).

4.3.1 Service description

Listing 1 shows the service definition section for security monitoring SLAs. Four security monitoring services are defined: these are services to monitor vulnerabilities in Mysql, Apache, InstaLinker and Custom Contact Form. Each service contains one vulnerability except Apache, which is monitored for two vulnerabilities. The services are running in three different servers. The infrastructure section shows detail about the servers. A Web server, database server and

content management server are used. According to the SLA, Apache is running on the web server, Mysql is running on the database server and WordPress plug-ins are running on the content management server.

```
<cloudService>
<macro>
<securityMonitoring id="Mysql-SM-ID" description="service definition for brute force login monitoring">
  <software id="Mysql-ID" name="Mysql" version="14.14"
       distribution="5.6.31" license="GPL" mode="mode" />
  <vulnerabilities>
    <vulnerability id="Mysql-V-1" name="Brute force access" cve_Id=""
            description="A token comparison based authentication is vulnerable to attacks of guessing (an attacker can try as many times as possible to find the correct token)" />
  </vulnerabilities>
  <infrastructure id="DBserver-ID" description="Database server ">
    <compute id="DB-VM-1" name="DBserver" architecture="" hostname="" cores="" speed="" memory="" />
  </infrastructure>
</securityMonitoring>

<securityMonitoring id="Apache-SM-ID" description="">
  <software id="Apache-ID" name="Apache" version="4.4.11"
       distribution="" license=" Apache License" mode="mode" />
  <vulnerabilities>
    <vulnerability id="Apache-V-1" name="Port scanning" cve_Id=""
             description="scanning ports to detect available services on each port" />
    <vulnerability id="Apache-V-2" name="Denial of service" cve_Id=""
            description="congesting a network to alter the availability of a service" />
  </vulnerabilities>
  <infrastructure id="Websever-ID" description="Web server ">
    <compute id="WEB-VM-1" name="Webserver" architecture="" hostname="" cores="" speed="" memory="" />
  </infrastructure>
</securityMonitoring>

<securityMonitoring id="IL-SM-ID" description="">
  <software id="IL-ID" name="InstaLinker" version=" &lt;= 1.1.1"
       distribution="" license="GPLv2" mode="mode" />
  <vulnerabilities>
    <vulnerability id="IL-V-1" name="Cross-Site Scripting (XSS)" cve_Id="8382 in WPVDB_ID"
         description="Due to a lack of input sanitization in some file, it is possible to utilise a reflected XSS vector to run a script in the target user's browser and potentially compromise the WordPress installation." />
  </vulnerabilities>
  <infrastructure id="CM-ID" description="Content managnment server ">
    <compute id="CM-VM-1" name="Cmserver" architecture="" hostname="" cores="" speed="" memory="" />
  </infrastructure>
</securityMonitoring>

<securityMonitoring id="CCF-SM-ID" description="">
  <software id="CCF-ID" name="Custom Contact Forms" version=" &lt;= 5.0.0.1"
       distribution="" license="GPLv2" mode="mode" />
  <vulnerabilities>
    <vulnerability id="IL-V-1" name="SQL injection" cve_Id="7542 in WPVDB_ID" description="
         unauthenticated users to download a SQL dump of the plugins database tables. It's also possible to upload files containing SQL statements which will be executed." />
  </vulnerabilities>
  <infrastructure id="CM-ID" description="Content managnment server ">
```

```
47      <compute id="CM-VM-1" name="Cmserver" architecture="" hostname="" cores="" speed="" memory=""
            />
48    </infrastructure>
49  </securityMonitoring>
50
51  </macro>
52  </cloudService>
53  </cloudServices>
```

Listing 1: Security monitoring service description in ECSLA

Lines (3 -13) define monitoring of a brute force login in Mysql version 14.14 database. Lines (15 - 27) define a service for Apache to be monitored against port scanning and DoS. Lines (29 - 38) and (40 - 49) define security monitoring services for InstaLinker and Custom Contact Forms respectively.

4.3.2 Parameters

Listing 2 shows a list of parameters that are used to define the metrics, monitoring, and schedule. The SLO can be verified according to these definitions. Four simple metrics namely *TP, FP, TN, FN*, and a complex metric C_{ID} are defined as parameters, lines (2 - 24). For the formulation of C_{ID} see [15]. In practice, the service provider can build a resource file describing a metric computation process, and the SLA definition can refer to this document for metric computation. Such practice (referring to other official documents for support) is not uncommon. For example, currently Amazon SLA [16] refers to a customer agreement document to exclude some cases from the SLA. The Parameter section also specifies monitoring and schedule, lines (26 - 27). Our previous study [1] discusses how to choose the timing for verification. However, it is important to note that this is an agreement: every part can be negotiated and is set to values satisfying the participants.

In our example, the verification can be done three times in 24 hrs, and the minimum value should satisfy the expected SLO. The example also specifies a schedule with a start and end time to perform verification.

```
1   <parameters>
2     <metric id="TP" name="True positive" unit="count" type="simple">
3       <description description="The number of correctly detected attacks"> </description>
4     </metric>
5
6     <metric id="FP" name="False positive" unit="count" type="simple">
7       <description description="The number of legitimate inputs mistakenly classified as attacks">
            </description>
8     </metric>
9
10    <metric id="FN" name="False negative" unit="count" type="simple">
11      <description description="The number of attacks that are not detected"> </description>
12    </metric>
13
14    <metric id="TN" name="True negative" unit="count" type="simple">
15      <description description="The number of legitimate that are classified as legitimate"> </
            description>
16    </metric>
17
18
19    <metric id="CID" name="Intrusion Detection Capability" unit="" type="complex">
20      <description description="The ratio of the mutual information between input and output to the
            entropy of the input
21                  (Single metrics to describe the effectiveness of an NIDS, computed from TPR,
                        FPR and B)">
```

```
22        </description>
23        <formulation> As described in Section 2.4.4 </formulation>
24    </metric>
25
26    <monitoring id ="Mon-1" statistic="min" window="24 hrs" frequency="3"/>
27    <schedule id ="Sch-1" start="8:00pm" end="8:00am"/>
28  </parameters>
```

Listing 2: Example parameters in ECSLA

4.3.3 Guarantees

Listing 3 shows the guarantee part which contains three sections. The *scope*, lines (3 - 8), shows services that are covered under this guarantee. In our example, the guarantee addresses all the four services. The *requirements* section, lines (10 - 14), defines base rate boundaries. The guarantee is for base rate values greater than 10^{-7} and less than 10^{-1}. If the base rate is not in this range, the SLO may not be achieved and such incidents are not SLO violations. The *terms* section, lines (16 - 31), contains only one term, a term describing expected C_{ID} value. It describes that the expected C_{ID} value should be computed using the formula presented in Section 5. The next section describes why a formula is given, rather than an actual C_{ID} value. The section also presents the process used for generating such a formula. To show an example usage for such a formula, let us assume the NIDS is evaluated (or an attack occurs) with a base rate value of 7×10^{-2}, putting this value into the formula gives an estimated value of $TPR \approx 0.71046$ and $FPR \approx 0$, and from these values we can compute the expected C_{ID} value, $C_{ID} \approx 0.7162$.

The fuzziness value of 0.05 (5%) and confidence ratio of 95% are also defined in the term section. This is interpreted as: from 100 verification tests with a base rate value of 7×10^{-2}, in at least 95 of the tests, the configured NIDS must perform with $C_{ID} >= 0.7162$ and the remaining tests must perform with $C_{ID} <= 0.7162$ but $C_{ID} >= 0.6562$. As a result of fuzziness and confidence ratio, 5% of the verification tests are allowed to perform below the guaranteed performance level without violating the SLO. These margins allow for the SLO to fluctuate to some extent. This is an exciting feature, especially for security monitoring SLO because it gives some level of freedom in achieving the expected SLO. In addition, the monitoring frequency and schedule, which are a reference to the parameter section, are defined in the terms section.

```
1  <guarantees>
2    <guarantee id="G-1">
3      <scope id="Sc1">
4        <service id="Mysql-SM-ID" subid="Mysql-SM-ID-mode"/>
5        <service id="Apache-SM-ID" subid="Mysql-SM-ID-mode"/>
6        <service id="IL-SM-ID" subid="Mysql-SM-ID-mode"/>
7        <service id="CCF-SM-ID" subid="Mysql-SM-ID-mode"/>
8      </scope>
9
10     <requirements>
11       <Requirement id="R1">
12         <Specification id="Sp1" policy="Required"> Base Rate(B), B >= 10^(-7) and B <= 0.1 </
               Specification>
13       </Requirement>
14     </requirements>
15
16     <terms>
17       <term id="T1" operator="">
18         <item id="CIDTerm"/>
19       </term>
20
21       <objective id="CIDTerm" priority="1" actor="provider">
```

```
22        <precondition policy="Required">
23          <description> The threshold value should be computed with function as described in
                Section 4.5
24          </description>
25        </precondition>
26
27        <expression metric="CID" comparator="gt" threshold="" unit="" monitoring="Mon-1" schedule="
              Sch-1"
28                confidence="95" fuzziness_value="0,05" fuzziness_percentage="5"/>
29      </objective>
30
31    </terms>
32  </guarantee>
33 </guarantees>
```

Listing 3: Example guarantee in ECSLA

By now it should be clear that we are using the C_{ID} metric to describe the performance of an NIDS in SLAs. However, the guarantee definition contains a function (a model) to calculate the expected C_{ID} value rather than an actual number. The reason is that it is not possible to know how often attacks will occur i.e it is impossible to know the base rate value before the occurrence of attacks, hence we cannot calculate the C_{ID} value in advance while defining an SLA. Moreover, using a single base rate value in the SLA definition makes the SLA very strict, i.e the SLO will be achieved if attacks occur only with that specific rate. Using a model allows estimating the expected performance for previously unknown base rate values. Additionally, if there are cases where the model cannot predict well, the cases can be categorically excluded from the SLA. For example, in Listing 3 $B < 10^{-7}$ and $B > 10^{-1}$ are excluded.

5 Including Unknown Base Rate Values in SLO

In this section, we present a method used to generate a model. The SLA definition uses the model to guarantee the performance of a monitoring device. The model helps to make the SLA inclusive, i.e. to give guarantees even for previously unknown base rate values.

Axelsson [5] presented the effect of base rate fallacy problem for NIDSs and showed the importance of the base rate in evaluating the performance of NIDSs. The base rate measures the prior probability of intrusion in the input data examined by the NIDS. Predicting or calculating the exact value of the base rate in advance is difficult if not impossible. As a result, the SLA definition will not include base rate values to describe the performance of security monitoring devices. Instead, the SLA will contain function(s) which takes the base rate as an input to calculate the expected TPR and FPR values. This helps to avoid an SLO which is defined for a specific base rate and be respected if an attack happens only on that rate.

As described in [15], C_{ID} is a function of TPR, FPR, and B. Any factor that can change the values of TPR and FPR indicates an effect on the performance of the NIDS. The factors affecting the performance of an NIDS can be grouped into two categories. First, *external factors* which includes mainly the rate of the inputs (throughput), the base rate and available resources for the NIDS. Second, *internal factors* which includes mainly the number of rules and the number of services that are configured to be monitored by that NIDS. Before describing the model generation method, we present the assumptions and challenges for designing such a method.

Term	Proba. Representation	Description
FPR (α)	$P(A\|\neg I)$	The probability that there is an alert, when there is no intrusion
TNR ($1-\alpha$)	$P(\neg A\|\neg I)$	The probability that there is no alert, when there is no intrusion
FNR (β)	$P(\neg A\|I)$	The probability that there is no alert, when there is intrusion
TPR ($1-\beta$)	$P(A\|I)$	The probability that there is an alert, when there is intrusion

Table 2: NIDS metrics with probabilistic description

5.1 Assumptions and Challenges

If we are given a fixed number of services and vulnerabilities to be monitored, it is safe to assume that the internal factors affecting the performance of the NIDS are constant values. Indeed, since we have a fixed number of vulnerabilities to be monitored, we assume that the rules configured in an NIDS are constant. In addition, we assume there are enough resources that are required by the NIDS to perform the monitoring task. Hence, the change in the rate of the inputs (throughput) will not affect the performance of the NIDS by creating resource scarcity. The remaining factor affecting the performance of the NIDS is the attack rate or base rate.

The four basic metrics that can be counted from the NIDS output are (TP, TN, FP, FN). Using these basic metrics, we can calculate TPR, FPR, TNR, and FNR. These values can be described in probabilistic terms as shown in Table 2. Using these values and the base rate, C_{ID} can be computed as described in [15].

In realistic production sites the value of the base rate is very small (i.e close to zero). That means an attack packet happens very rarely compared to legitimate packets. Gu *et al* [15] assumed base rates in the range of $\{1 \times 10^{-2} - 1 \times 10^{-6}\}$. In 1999 Axelsson [5] supposed that the maximum value was $\{2 \times 10^{-5}\}$ (2 intrusions per day from $1,000,000$ records, an intrusion affecting 10 records in average).

While doing an experiment, achieving such a very low base rate is challenging and it requires an enormous amount of resources. In our work, we evaluate an NIDS by injecting real attacks. The actual attack injection algorithm which takes the base rate as an input is presented in our previous study [1]. Here we present the challenges in a high-level description. While performing the attack injection, packets and related information are logged and analyzed later with the output of NIDS for metrics computation. To achieve low base rates, attacks are injected very rarely and to have a statistically sound result the injection is performed not only once but multiple times. Hence, the experiment takes a long time. In addition, logging packets for such a long time requires large disk space. The challenge exacerbates when assuming the experiment needs to be performed for a large number of vulnerabilities.

Additionally, we assume the SLA specifies the lowest guaranteed base rate value. i.e SLA will not be violated for attack occurrences below the specified base rate. However, such lower bound values should be as realistic as possible. Finally, we assume the provider can perform the evaluation at the lowest guaranteed base rate value at least once. Section 7.4 describes the drawbacks of our estimation as a result of these assumptions.

5.2 Metrics Estimation Method

Figure 3 shows the correlation between B and C_{ID} for different TPR and FPR values. The plot assumes TPR and FPR as constant values. A close look at Figure 3 shows a trend between the value of C_{ID} and different B values. Nevertheless, in practice configuring an NIDS to generate constant TPR and FPR values is difficult. This is because in a realistic operational environment the values of TPR and FPR are affected not only by its internal configuration but

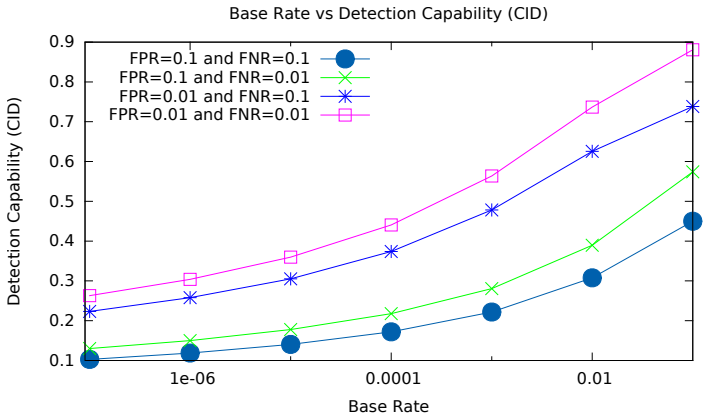


Figure 3: Base rate (B) vs C_{ID}

also by external non-deterministic factors, e.g. the base rate.

Following this observation, we performed experiments to identify if there is a correlation between B and (TPR, FPR). If such correlation exists, we can have a model showing the correlation from known values of B and (TPR, FPR), then we can use that model to estimate TPR and FPR values for unknown base rates. The estimated values can be used to compute C_{ID} for those unknown base rates.

To generate the model we propose an interpolation-based method. The method takes known values of TPR and FPR which are computed on different B values. These values are used to generate a fitting function that can approximate the points. The generated function(s) takes B as an input and produces TPR and/or FPR values. The function is used in SLAs to estimate the TPR and FPR values for previously unknown B value. In practice, such an estimation method may not provide the exact value, but it gives an approximation to the exact value. Moreover, the margins described in ECSLA (fuzziness and confidence ratio) helps to tolerate some degree of variation in the SLO. Given a security monitoring configuration, to generate a representative formula which can be used to estimate the TPR and FPR values, we follow the following steps:

- Compute the performance of the NIDS on a given configuration taking a known base rate value as an input. The procedure to do the evaluation is presented in our previous study [1]. The performance evaluation should include the lowest guaranteed base rate and as many other points as possible. Increasing the number of evaluation points results in an increase in the accuracy of the model.

- Using results from the performance test, find a correlation (function f) between TPR and/or FPR and B, i.e. f is a function of B and evaluating f at some B value produces TPR and/or FPR.

This way we can generate an equation to be used in the SLA definition. The function f can be used to estimate the expected TPR and FPR values for new B values. The function f may not represent the exact relationship, but it is derived from the best information available. Section 7.1 presents an experimental evaluation showing the feasibility of our metrics estimation method.

6 NIDS Performance with a Large Number of Vulnerabilities

By now it should be clear that we aim at defining SLAs to guarantee the performance of signature-based NIDSs. Service providers need to prepare SLA templates that will be offered to potential tenants at the start of the negotiation process. To achieve this, service providers need to build a knowledge base on the performance of their security monitoring ability. As described in Section 2.2 one of the challenges to prepare SLO templates is the fact that there are thousands of vulnerabilities and a tenant may choose any combination of those vulnerabilities. That means a service provider needs to prepare SLA templates based on all combinations of vulnerabilities. This is not practical as it requires the service providers to perform a huge amount of tests just to prepare template SLOs.

Let us assume there are n vulnerabilities that a service provider can monitor and offer to provide security monitoring service. A tenant is interested in k of these vulnerabilities, where usually $k \ll n$. To prepare an SLO template based on every combination means to prepare an SLO for a combination of n and k, $\binom{n}{k}$ for all k in (1 ... n). For example, if we have a thousand vulnerabilities, preparing an SLO for every combination of vulnerabilities require more than tens of millions of operations. This is very tedious and not efficient.

In addition, the number of services that are monitored under a given security monitoring configuration affects the effectiveness of the monitoring process. This consequence is intuitive, as having more services to monitor means having additional tasks and more input, hence there is an effect on the performance of the monitoring device. If we retake the above example, let us assume there are two tenants, the first one selects k vulnerabilities and the second one selects m vulnerabilities where $m > k$ and $k \in m$ (m includes all k vulnerabilities). The security monitoring service will perform better for the first tenant. The second tenant will have more tasks to be done; as a result, the performance will degrade.

In practice, NIDSs use rule(s) to monitor a vulnerability and more vulnerabilities means more rules to be added in NIDS. Thus, the inputs will be evaluated against more rules. We refer to the effect of a rule on other rules as *interference*, i.e when a rule interferes with the functioning of another rule.

Quantifying the interference is useful to compute the performance of an NIDS which is configured with more than one rule. Given an NIDS with such a configuration, i.e. an NIDS configured with more than one rule, we propose a performance evaluation method which uses the interference value between rules. Let us assume the NIDS is configured with n rules. For simplicity, we assume one rule is used to monitor a single vulnerability. For a given rule, the collection of the interferences between that rule and the remaining other $(n-1)$ rules form a *vector*. For all n rules, the collection of their interference vector forms a *matrix*. However, in practice, building such vectors and matrices requires a large number of computations. To reduce the required number of computations, we propose a clustering method which groups rules based on a given criterion. A formal description of interferences is presented in the next section.

In the next sections we address the problems described above, i.e we want to answer questions like, can we model and evaluate the effect of having more vulnerabilities, and thus more interferences between rules? Can we provide a better way for service providers to build a knowledge base while having a large number of vulnerabilities?

6.1 Modeling Rule Interference and its Effect on NIDS Performance

In this section, we want to address the problem of interference between vulnerabilities. Understanding the effect of a vulnerability on other vulnerabilities is essential in order to prepare

custom-tailored SLA templates efficiently. To show the importance of measuring interference, let us assume we have n vulnerabilities and corresponding rules used in an NIDS. The performance of an NIDS varies depending on the number of configured rules. Assuming all other factors affecting the performance of an NIDS are constant, the performance and number of rules are inversely proportional, i.e the performance decreases as the number of rules increases. Moreover, it is not enough to know the existence of such interferences between vulnerabilities: we need to measure and *quantify* the level of interference in order to use it in the process of SLA definition. If a tenant selects k vulnerabilities out of n, the SLO offered for that tenant is the performance of the NIDS regarding those k vulnerabilities.

While selecting k out of n vulnerabilities, if we don't have a quantitative measure of the interference between the k vulnerabilities (i.e between the rules to monitor those vulnerabilities) then the provider is expected to run a test in order to measure the performance of its NIDS on k vulnerabilities. This process happens for every possible set of k rules. Having quantitative knowledge of the interference helps to do statical analysis rather than running a dynamic test. Therefore, it reduces the number of performance tests to prepare SLO templates. We present a formal definition of the interference and how to perform the static analysis in the next section.

6.2 Rule Interference in NIDS

As described above, the interference between rules refers to the effect of one rule on another rule (alternatively, it can be seen as the effect of a vulnerability on another vulnerability, assuming a single rule is used to monitor a single vulnerability). Before formally defining the interference between rules, we describe the assumptions taken in the formulation of the interference.

6.2.1 Assumptions

NIDSs take rules as an input. The rules are mechanisms to tell the NIDS what to look in the input packets. The detection engine of an NIDS applies the rules on each packet. If the packet matches a rule, the specified action of that rule is taken, and log(s) and/or alert(s) will be generated. However, if the packet matches with more than one rule, the NIDS behaves in one of the following ways:

- It generates an alert for all the matchings ;
- It generates an alert for few of the matchings based on some heuristics (e.g. the first match, the most severe).

If the NIDS follows the second option, in some cases it may stop further processing after finding the heuristic (e.g. after finding the first match). Usually, the second option is the default property, but even if it is not optimal, NIDSs can be configured to behave like the first option. While formally defining rule interferences we assume an NIDS configured to generate an alert for all matchings (the first case). i.e. if a packet matches with more than one rule, it generates an alert for every match. Assuming this we define interference as follows.

6.2.2 Formal definition of interference

Let us assume we have two vulnerabilities to be monitored (V_i and V_j) with a set of rules for each of them to be configured in an NIDS. For a given base rate (B), an NIDS configured to monitor V_i will generate (TP_i, FP_i, TN_i and FN_i) the same for V_j, it generates (TP_j, FP_j, TN_j and FN_j). These are the basic metrics and they are measured by counting the number of attacks

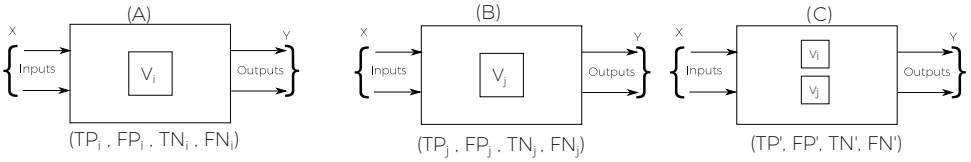


Figure 4: NIDS outputs

(or legitimate traffic) that are correctly (or incorrectly) classified by the NIDS from a given input. From these basic metrics we can calculate TPR and FPR as:

$$TPR_i = \frac{TP_i}{TP_i + FN_i} \ , \ FPR_i = \frac{FP_i}{FP_i + TN_i} \ , \ TPR_j = \frac{TP_j}{TP_j + FN_j} \ and \ FPR_j = \frac{FP_j}{FP_j + TN_j} \tag{1}$$

Interference between V_i and V_j is an event that can happen when both are configured together in one NIDS. Figure 4 (A and B) shows a box diagram, representation of an NIDS configured to monitor V_i and V_j separately. Moreover, Figure 4(C) shows an NIDS configured to monitor both V_i and V_j. As we have NIDS outputs for V_i and V_j separately, an NIDS configured to monitor both vulnerabilities together generates (TP', FP', TN' and FN').

We define the interference as the change in values between TP' and $TP_i + TP_j$ (respectively FN' and $FN_i + FN_j$) and between FP' and $FP_i + FP_j$ (respectively TN' and $TN_i + TN_j$). Formally we can define an interference as follows:

Definition 6.1. There is an interference between two vulnerabilities V_i and V_j if an NIDS configured with V_i gives a quadruple value (TP_i, FP_i, TN_i and FN_i), an NIDS configured with V_j gives a quadruple value (TP_j, FP_j, TN_j and FN_j), an NIDS configure to monitor both V_i and V_j gives a quadruple value (TP', FP', TN' and FN') and

$$TP' \neq TP_i + TP_j \ \ or \ \ FP' \neq FP_i + FP_j \ \ or \ \ TN' \neq TN_i + TN_j \ \ or \ \ FN' \neq FN_i + FN_j \tag{2}$$

In other words, if an NIDS is configured to monitor both vulnerabilities (V_i & V_j) and the output is not equal to the sum of separate outputs from V_i and V_j then we can say there is an interference. Note that interference is just the change in values it can be an increase or decrease of (TP', FP', TN' and FN') from the sum of (TP_i, FP_i, TN_i, and FN_i) and (TP_j, FP_j, TN_j, and FN_j). However, an increase in TP' or TN' is a *positive interference*. It indicates better effectiveness of an NIDS and such type of changes increases the C_{ID} value. It is unlikely for this event to happen.

Therefore, we are interested in *negative interferences* which indicate a degradation in the effectiveness of an NIDS and decrease the C_{ID} value. In other words, an increase either in the value of FP or FN means an increase of false classification or errors made by the NIDS. We can represent the interference between V_i and V_j as $(FP, FN)_{ij}$. The change in FP and FN can be described as:

$$FP_{ij} = FP' - (FP_i + FP_j) \ \ and \ \ FN_{ij} = FN' - (FN_i + FN_j) \tag{3}$$

Moreover, (TP', FP', TN', and FN') can be expressed as follows. Note that an increase in

Vule.	V_1	V_2	V_3	.....	V_n
V_1	0	$(FP, FN)_{1,2}$	$(FP, FN)_{1,3}$	.....	$(FP, FN)_{1,n}$
V_2		0	$(FP, FN)_{2,3}$	.....	$(FP, FN)_{2,n}$
V_3			0	.....	$(FP, FN)_{3,n}$
.....				.....	.....
V_n				.....	0

Table 3: Interference Matrix (IM)

FP shows a decrease in TN and the same way and an increase in FN shows a decrease in TP.

$$\begin{aligned} TP' &= TP_i + TP_j - FN_{ij}, \;\; FN' = FN_i + FN_j + FN_{ij} \\ FP' &= FP_i + FP_j + FP_{ij}, \;\; TN' = TN_i + TN_j - FP_{ij} \end{aligned} \tag{4}$$

6.2.3 Interference vector (IV) and interference matrix (IM)

If there are n vulnerabilities, the interference between V_i and the other $n-1$ vulnerabilities can be described with an *interference vector (IV)*

$$\textbf{interference vector (IV),}\ V_i = \{(FP, FN)_{i1}, (FP, FN)_{i2}, ...(FP, FN)_{in}\}$$

Interference vectors of n vulnerabilities will form an *interference matrix* as shown in Table 3. An entry in the table shows an interference between the vulnerabilities in the corresponding column and row. The matrix is triangular and there is no interference for a vulnerability with itself.

Given such quantitative values of an interference, the next question is how we can compute the expected C_{ID} value for an NIDS configured to monitor both V_i and V_j? As we know C_{ID} is a function of TPR, FPR and B. Using Equation 1 to compute the TPR and FPR values for (TP', FP', TN', and FN') may not be possible if there is interference between the vulnerabilities V_i and V_j. The equation needs to take into account the changes created as a result of putting multiple vulnerabilities together. In the next section, we present how to compute aggregated metrics, TPR and FPR values while aggregating multiple vulnerabilities in a given NIDS.

6.2.4 Computing aggregated metrics

Assume we have two vulnerabilities V_i and V_j, and an NIDS configured to monitor both V_i and V_j. The TPR' and FPR' values for this NIDS is represented as TPR_{agg} and FPR_{agg}. They are values computed from (TP_i, FP_i, TN_i, and FN_i), (TP_j, FP_j, TN_j, and FN_j) and $(FP, FN)_{ij}$.

If there is no interference between the two vulnerabilities, that means there is no change between the output of the NIDS and the sum of the separate outputs from V_i and V_j (i.e $(FP, FN)_{ij} = (0, 0)_{ij}$). From Equation 4 we have

$$TP' = TP_i + TP_j, \;\; FN' = FN_i + FN_j, \;\; FP' = FP_i + FP_j \;\; and \;\; TN' = TN_i + TN_j \tag{5}$$

TPR_{agg} and FPR_{agg} can be computed as follows:

$$\begin{aligned} TPR_{agg} &= \frac{TP'}{TP' + FN'} = \frac{TP_i + TP_j}{TP_i + TP_j + FN_i + FN_j} \quad and \\ FPR_{agg} &= \frac{TP'}{TP' + FN'} = \frac{FP_i + FP_j}{FP_i + FP_j + TN_i + TN_j} \end{aligned} \tag{6}$$

A TPR is computed as the ratio between *correctly detected attacks* and the *total number of attacks*. Equation 6 can be interpreted as, since there is no interference between V_i and V_j the number of correctly detected attacks are the sum of TPs from V_i and V_j. The total number of attacks is the sum of attacks from V_i and V_j, i.e let us assume we use the packet flow f_i to evaluate the NIDS configured with V_i which contains a total number of $(TP_i + FN_i)$ attacks. Also, we use the flow f_j to evaluate the NIDS configured with V_j which contains a total number of $(TP_j + FN_j)$ attacks. Then to evaluate the NIDS configured with both V_i and V_j we use both f_i and f_j which results in a total number of $(TP_i + FN_i + TP_j + FN_j)$ attacks.

To keep the evaluation consistent f_i and f_j are the same. However, the flow contains packets that can trigger alerts for both of the vulnerabilities. Also, in some cases a packet may match more than one rule; as a result, the NIDS generates alerts for every match, as described in Section 6.2.1. Equation 6 can be generalized for n vulnerabilities as:

$$TPR_{agg} = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FN_i} \quad and \quad FPR_{agg} = \frac{\sum_{i=1}^{n} FP_i}{\sum_{i=1}^{n} FP_i + \sum_{i=1}^{n} TN_i} \tag{7}$$

If there is an interference between V_i and V_j then there is a difference between the output of an NIDS configured to monitor both V_i and V_j and the sum of the separate outputs from V_i and V_j (i.e $(FP, FN)_{ij} \neq (0, 0)_{ij}$). Taking Equation 4, we can calculate TPR_{agg} and FPR_{agg} as follows:

$$\begin{aligned} TPR_{agg} &= \frac{TP'}{TP' + FN'} = \frac{TP_i + TP_j - FN_{ij}}{TP_i + TP_j + FN_i + FN_j} \quad and \\ FPR_{agg} &= \frac{FP'}{FP' + TN'} = \frac{FP_i + FP_j + FP_{ij}}{FP_i + FP_j + TN_i + TN_j} \end{aligned} \tag{8}$$

If we have n vulnerabilities, we can generalize TPR_{agg} and FPR_{agg} as shown in Equation 9. In the equation, in addition to the assumptions described above, we assume that interferences are limited to pairwise interferences, i.e. the only interferences in the group are than the ones in the pairs of rules.

$$TPR_{agg} = \frac{\sum_{i=1}^{n} TP_i - \sum_{i=1,j=1}^{n} FN_{ij}}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FN_i} \quad and \quad FPR_{agg} = \frac{\sum_{i=1}^{n} FP_i + \sum_{i=1,j=1}^{n} FP_{ij}}{\sum_{i=1}^{n} FP_i + \sum_{i=1}^{n} TN_i} \tag{9}$$

Given the performance of an NIDS on n vulnerabilities separately and the interference between them for some base rate value B, we can compute the expected C_{ID} value for that NIDS by doing only static analysis. This is done by computing TPR_{agg} and FPR_{agg} as shown above and use those values with B to calculate C_{ID}.

It is important to note that, we still have the issue with a large number of vulnerabilities. The method described above may reduce the number of required tests to prepare SLO templates

to some degree, but it still requires to perform at least a test for every couple of vulnerabilities, i.e if we have n vulnerabilities the above method requires at least $\binom{n}{2}$ (combination of n and 2) performance tests. In practice when n is very large, the method may not be practical.

In the next section, we present our proposed solution to reduce the required number of evaluations in order to prepare SLO templates. Our goal is to have a knowledge base on the performance of an NIDS in order to prepare template SLAs and offer the templates to a potential tenant.

6.3 Building a Knowledge Base by Clustering Vulnerabilities

In order to realize a security monitoring SLA, the conflict between having a large number of vulnerabilities and the need to have custom tailored SLOs needs to be solved. Otherwise, running numerous performance tests just for building a knowledge base is not practical. Moreover, this process is not a one time task: it may be required to do the test on different occasions. For example, as new vulnerabilities are discovered the NIDS needs to be tested on those vulnerabilities. Hence, having an efficient method, which can reduce the required number of performance tests significantly affects the practicality of a security monitoring SLAs. In order to achieve this, we propose a method based on clustering vulnerabilities.

The idea is to perform the evaluation test per group rather than for each vulnerability. The clustering mechanism is applied to the NIDS rules based on heuristics, and the resulting groups are used to build a knowledge base for the performance of NIDSs. Clustering, as the name indicates, is an act of grouping elements to some kind of classes called *clusters* (see Figure 5). After constructing such groups the evaluation can be done per group, i.e configuring an NIDS to monitor all vulnerabilities in one group and measure the performance. If the interference between the vulnerabilities in a group is a *negative interference*, then the result of this evaluation is a *lower bound* for the performance of an NIDS configured with any subset of the group. However, even if the chance of having a *positive interference* is very small, the provider should be careful while setting the lower bound values. If the interference in a group is a *positive interference*, it may not represent a lower bound for a subset of the vulnerabilities in that group.

Let us take for example that we have one hundred vulnerabilities, and by clustering, we formed ten groups $(G_0, G_1 ... G_9)$ each G_i containing ten vulnerabilities $(V_0, V_1 ... V_9)$. Assume an NIDS is configured to monitor all the vulnerabilities in G_i. Let us say evaluating the performance of that NIDS for a given base rate value B results in C_{ID} value of x. If the interference between the vulnerabilities in G_i is a negative interference, then we can say that x is a lower bound on the performance of the NIDS for any subset of vulnerabilities from G_i. This is following Equation 9, assuming negative interferences, the FPR_{agg} for all V_is in G_i is higher than V_is for any subset in G_i. Hence, the C_{ID} will be the lower bound while taking all vulnerabilities in G_i. However, if the interference between the vulnerabilities in G_i is a positive interference, we can not set x as a lower bound for that group. In other words, if the interference is positive, then there may be a subset of vulnerabilities that could result in a worse performance than putting all the V_is in G_i together. It is unlikely for such an event to happen, but it is essential to consider the case while drafting an SLO.

Subsequently, Table 3 will have a smaller height and width. The columns and rows will represent groups of vulnerabilities (G_i) instead of single vulnerabilities (V_i). Service providers can choose appropriate heuristics to group vulnerabilities. Some examples of heuristics are described below. These are example heuristics, a provider needs to consider the available resources and the grouping criteria. Having a minimal number of groups results in meager SLO offer and having a large number of groups requires huge amounts of computations. Providers need to select the

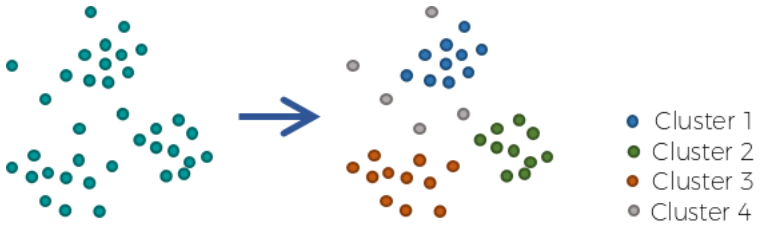


Figure 5: Clustering NIDS rules

Groups	G_1	G_2	.....	G_n
G_1	...	...	.....	...
G_2	...	...	.....	...
...	...	...	.....	...
G_n	...	...	.....	...

Groups	G'_1	G'_2	.....	G'_n
G'_1	...	...	.....	...
G'_2	...	...	.....	...
...	...	...	.....	...
G'_n	...	...	.....	...

Table 4: Multiple interference matrix from groups using different heuristics

heuristics considering available resources. Some examples of heuristics include:

- Grouping vulnerabilities based on related application, for example, OS vulnerabilities (vulnerabilities to monitor different OSs), browser vulnerabilities, ...
- Grouping vulnerabilities based on related applications that are used together, for example, grouping vulnerabilities in the LAMP stack (Linux, Apache, Mysql, and PHP) applications in one group,
- Grouping based on the nodes and services they are providing or used for, e.g. grouping vulnerabilities for login nodes together, storage nodes,
- Grouping based on the threat or severity of the vulnerabilities, for example, grouping less severe vulnerabilities together.

Having such groups and a minimized interference matrix, it is possible to compute a minimum expected performance for any given vulnerability. When a tenant needs to be monitored and selects the vulnerabilities from these groups, there are two cases. First, if the selected vulnerabilities are in the same group, then the performance of the NIDS on that group is offered as an SLO. Second, if the selected vulnerabilities are in different groups, then we can calculate the expected C_{ID} value using the method described in the previous section.

It is possible to group vulnerabilities based on multiple heuristics. Each heuristic produces a table as shown in Table 4. For a given set of vulnerabilities, the table which produces a better performance can be used to offer the SLOs.

In the next section, we present evaluations performed to validate the ideas proposed in this paper.

7 Evaluation

We performed experiments to validate the proposed solution for the metrics estimation and rule clustering methods. The actual procedure for the experiment is presented in our previous

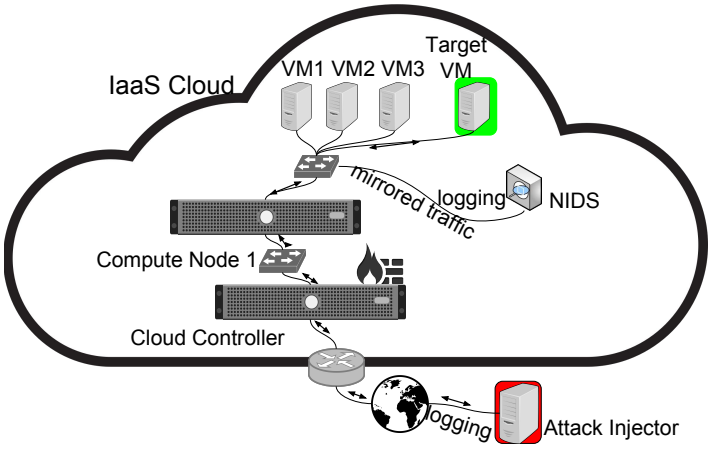


Figure 6: Experimental setup

study [1]. In this section, we present the setup and results from the experiment. The results show the basic metrics (TP, FN, FP, TN) of a given NIDS while configured to monitor a list of services. The results are shown for different base rate values. We also present how we managed to group NIDS rules based on some heuristics.

We aim to validate the C_{ID} approximation for unknown base rates and the efficiency of using the interference matrices, i.e to validate the efficiency of our clustering method to reduce the number of measurements required when preparing SLO templates. To make our experiment as realistic as possible, we perform a dynamic, real attack injection. By injecting real attacks we can see how a given NIDS performs and behaves on a specific configuration. This measuring technique is used to verify the correctness of an NIDS configuration in our previous study. Full details of the procedures are presented in [1]. Here we present an overview of the experiment setup and the result, counted as basic metrics, to show the process of SLO template creation.

7.1 Experimental Setup

Grid5000 [17] testbed infrastructure is used to run our experiments. We build a cloud infrastructure using OpenStack [18] and Open vSwitch(OvS) [19] as a virtual switch. Figure 6 shows a high-level architecture of the experimental setup. The tenant infrastructure is configured to run the services described in Table 1. For the experiment, three production virtual machines (VMs) are running the Apache, Mysql server and WordPress content management system. A fourth VM is also instantiated and used as a *target* for the injected attacks. The target VM exhibits similar properties as the production VMs by running all the three services. The virtual switch is configured to forward all attack packets only to the target VM. Hence, the attacks are prevented from disrupting the production VMs.

Snort [20], the most widely deployed IDS, is used as NIDS. It is deployed on a separate physical node, and it is connected to the virtual switch through a mirror port. All packets passing through that switch are also mirrored to the Snort node for analysis. Snort is configured to monitor the services listed in Table 1 i.e to look for an instance of attacks listed in the table and to output an alert for each matching rule. An attacker machine is located outside of the cloud and it is used to inject attacks i.e. to perform a dynamic attack injection campaign. We

	B	TPR	FPR	TNR	FNR	CID
Small B values	0.001	0.7005733333	0.000035352	0.9999566667	0.2994266667	0.6453533333
	0.003	0.7022433333	1.23E-07	0.9999966667	0.2977566667	0.6493066667
	0.005	0.7167533333	0.00000026	0.9999966667	0.26614	0.66004
	0.007	0.7278733333	8.63E-07	0.9999966667	0.2721233333	0.66845
	0.009	0.7567866667	0	1	0.2432133333	0.6968933333
	0.01	0.7338044444	8.12E-07	0.999996667	0.23843	0.700916667
Slightly higher B values	0.06	0.80297	0.001438933	0.99856	0.197023333	0.704433333
	0.07	0.80046	0	1	0.1995366667	0.71623
	0.08	0.8076033333	6.53E-06	0.9999966667	0.19239	0.73025
	0.09	0.8198066667	1.59E-05	0.9999833333	0.18019	0.76141
	0.1	0.8661833333	0.0009814231	0.9990166667	0.13381	0.7737933333

Table 5: TPR, FPR, TNR, FNR values of an NIDS from our experiment and calculated C_{ID} value for varying B

use real attacks to exploit vulnerabilities in the target VM. The attacks are interlaced with legitimate traffic according to the base rate. In addition, we know the number of packets sent by each legitimate and attack request.

To measure the performance of a configured NIDS, we record all the communications between the attacker and target VM, i.e. we record all the injected packets. Snort gets all these packets as an input and generates an alert(s) when it detects an attack. At the end of the attack campaign, we get the output of the NIDS. Using recorded inputs and the output alerts, we count the number of inputs that are correctly (wrongly) classified as attacks (legitimate requests) by the NIDS. This way we can count the (TP, TN, FP, and FN) values. More details about the setup and its justification are presented in [1].

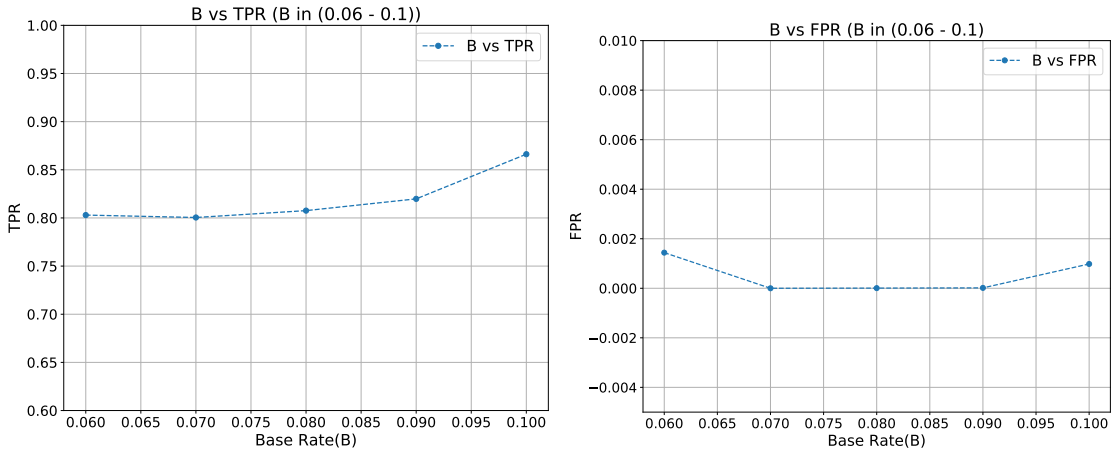
7.2 Collecting Data Points and Generating an Estimation Model

Using the setup described in the previous section we run experiments to measure the performance of the configured NIDS. Table 5 shows the *TPR,FPR,TNR*, and *FNR* values, these are calculated from basic metrics (TP, TN, FP, and FN). The table also shows the calculated C_{ID} value for a given B. The values shown are averages over three rounds for each computation.

The experiment is performed for two sets of base rate values, for the smaller B values in $(0.001 - 0.01)$ and a slightly higher B values in $(0.06 - 0.1)$. For this experiment, we assume an SLA which guarantees the performance of an NIDS with a lower bound base rate value of 10^{-3}, i.e the SLA will not be violated if the NIDS underperformed for an occurrence of attack with $B < 10^{-3}$. It is important to note that in practice this value (10^{-3}) is a relatively large lower bound, i.e. attacks usually occur with a base rate value $B < 10^{-3}$. As described in the previous section, the actual value of the base rate is very small. However, in our environment, $B = 10^{-3}$ is the lowest achievable base rate value. Section 7.4 presents a detailed explanation on the limitation of our experiment.

Figure 7 shows the plot of B vs *TPR* and B vs *FPR* for B in $(0.06 - 0.1)$. We use Table 5 and the corresponding graphs to model the relationship between B and (TPR, FPR). From the table and the plots, we observe that the FPR values are very small (close to zero). This is because the rules in the NIDS are carefully crafted for the set of considered attacks. As a result, false positives generated by these rules are very small. This property is consistent for different base rate values. Other studies [21] showed that given the attacks to be monitored, it is possible

to craft an NIDS rules with a minimal FP value. Taking these into consideration, we can use a constant $FPR = 0$ value in the SLA. However, for our SLA we take the highest FPR value, as C_{ID} is sensitive for changes in FPR. In the context of SLA, taking the highest FPR means promising a lower C_{ID} value, which puts the provider in a better position for not violating the SLA.



(a) B vs TPR (B in (0.06 - 0.1))

(b) B vs FPR (B in (0.06 - 0.1))

Figure 7: Plot of B vs TPR and B vs FPR for B in $(0.06 - 0.1)$

For the TPR, we can observe that its value is increasing with the B value. To model their relationship, we use the TPR values for B in $(0.06 - 0.1)$, and a TPR at the lowest guaranteed base rate, i.e $B = 10^{-3}$. We fit these points using a quadratic polynomial function, because there is only one local extremum value in the given range of B. The quadratic function f which best approximates these points is shown in Equation 10

$$f(B) = 0.7008827 + 1.357906 * B + 1.447843 * B^2 \tag{10}$$

Using this function, we can estimate the values of the expected TPR for other base rate values. As an example, Table 8 shows the estimated and actual TPR values for B in $(0.003 - 0.009)$. Figure 9 shows the plot of expected TPR in comparison with actual values. From this data, we can observe that our metric estimation method produce results which are close to the actual values.

Table 8 can be used as an input to drive the fuzziness and confidence ratio values. In addition to such a table, a provider may take other assumptions like the available resources to drive the appropriate fuzziness and confidence ratio values.

7.3 Clustering NIDS Rules

In Section 6.3 we presented a clustering technique that can be used to reduce the width and height of an interference matrix, thus reducing the number of required evaluation tests to prepare the template SLOs. In this section, we present an example using rules from Snort NIDS.

In the official snort rule repository, there are three categories of rule sets:

- Subscriber: The most updated set of rules which is available for paying customers and it is regularly updated.

B	Estimated TPR	Actual TPR
0.003	0.7049694	0.7022433333
0.005	0.7077084	0.7167533333
0.007	0.710459	0.7278733333
0.009	0.7132211	0.7567866667

Figure 8: Estimated and actual TPR values

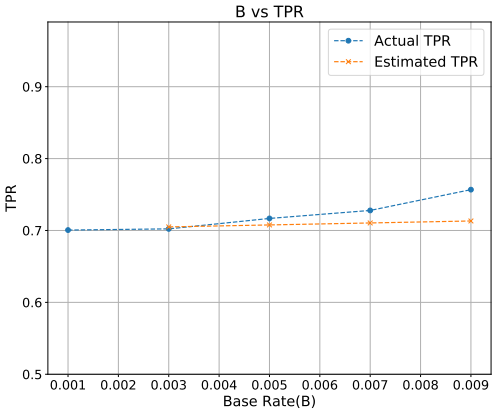


Figure 9: Plot of the estimated and actual TPR values

- Registered: This rule set is 30 days behind the Subscriber rule set and it is distributed free of charge.
- Community: A set of rules written by the community and verified by the company maintaining Snort. It is also distributed free of charge and it does not contain the subscribers rule set.

We used the registered rule set version 2.9.8.0 to demonstrate the clustering technique described above. It contains a total of 10628 rules in 53 files. We used several properties to group the rules including properties that are part of the rule syntax. The properties used include the *application type*, the *attack class type*, the *severity of the vulnerability*, and *applications working together*.

7.3.1 Grouping based on application type

Another method is grouping applications with a similar type of functionalities into the same group. For example, grouping vulnerabilities in the browsers or operating systems together in one group. By default, rules in the Snort repository are grouped per application, e.g. Firefox rules, Internet Explorer rules, office file rules, etc. This default grouping can be used for our clustering purpose and our grouping method reduces further the number of groups. Table 6 shows an example of grouping created using the application type.

7.3.2 Grouping based on attack type and severity

An attack type (class type) is part of the rule options used in Snort syntax. Rules in the Snort NIDS are divided into two sections, the *rule header* and *rule options*, where the header contains information like the rule's action, protocol, source and destination IP/port. The option contains the alert messages and information about which part of the packet should be used to generate the alert message.

The keyword '*classtype*' is used to indicate that a rule is detecting general kind of attack. We can use this keyword to classify NIDS rules for building the interference matrix. A few default *classtypes* are defined, but it is possible to define a custom *classtype* in the *classification.config* file. Table 7 shows classification of rules based on the *classtypes* property.

Application Group	Applications (number of rules)	Total
Browsers	Firefox (15), Internet explorer (1286), Webkit (2), Others (8), Plugins (31)	1339
Files	Executable (17), Flash (1506), image (125), java (110), multimedia (55), office (462), pdf (506), others (331), file-identifier (979)	4091
OS	Windows (388), Solaris (3), mobile os (3), others (36), netbios (22), Linux (15)	467
Protocols	imap (2), scada (10), telnet (3), dns (1), pop (1), snmp (1), voip (2), others(4)	24
Servers	Apache (27), mail (9), mysql (1), oracle (2), samba (6), web-apps(273), mssql(1), others (223)	542
Potentially Unwanted Application (PUA)	Adware(30), p2p(3), toolbars (4), others (21)	58
Malware	Backdoor (110), CNC (3053), malware-tools (14), others (336)	3513
Others	app-detector (2), exploit-kit (492), indicator-compromise (30), indicator-scan (2), policy-social (2), policy-others (15), sql (16), deleted(3), indicator-obfuscation (32)	594

Table 6: Rule classification based on application types

Similar to this, Snort rules have a *priority* parameter which allows defining the severity with integer values. A *priority* is also defined in *classification.config* file. For example, the default configuration contains four priorities. A priority of 1 (high) is the most severe and 4 (very low) is the least severe.

7.3.3 Grouping based on applications working together

It is very likely that an application is used in collaboration with other applications. To give a simple example, an operating system is used to run almost all applications. Such a relationship can be used to group vulnerabilities. Such grouping is natural, as applications are used together to provide the expected service. Moreover, it can result in a higher performance value, as aggregation from different groups will be minimized.

Examples of such method include grouping Linux, Apache, Mysql, PHP (LAMP) and WordPress in one group, Windows, Microsoft Office products and Internet Explorer in another group.

Using such grouping, we can reduce the number of required evaluations. If we take the example rule set to build the interference matrix using each rule, it requires more than 54 million tests (combination of 10628 by 2). Using the default grouping by application, it requires 1378 tests (combination of 53 by 2) and using application type requires 28 tests (combination of 8 by 2). It is important to note that there is a tradeoff between the number of required tests and the performance of an NIDS (or its C_{ID} value). Less number of groups means less number of tests, but it also produces smaller C_{ID} values than a large number of groups. This is because when having a small number of groups, each group will contain more vulnerabilities than when having a larger number of groups. The provider should take into account available resources for the tests and select the clustering method accordingly. Multiple grouping methods can be used to build multiple interference matrices (see Table 4) and offer SLOs from the one which produces the best result.

Class Type	Description
attempted-user (4332)	Attempted User Privilege Gain
protocol-command-decode (46)	Generic Protocol Command Decode
denial-of-service (38)	Detection of a Denial of Service Attack
default-login-attempt (2)	Attempt to login by a default username and password
misc-activity (1049)	Misc activity
suspicious-filename-detect (1)	A suspicious filename was detected
attempted-dos (117)	Attempted Denial of Service
attempted admin (728)	Attempted administrator privilege gain
trojan-activity(3904)	A network Trojan was detected
string-detect(1)	A suspicious string was detected
bad-unknown (3)	Potentially Bad Traffic
network-scan (1)	Detection of a Network Scan
misc-attack (37)	Misc Attack
attempted-recon (146)	Attempted Information Leak
policy-violation (57)	Potential Corporate Privacy Violation
successful-recon-limited (12)	Information Leak
unsuccessful-user (1)	Unsuccessful User Privilege Gain
web-application-attack (150)	Web Application Attack
successful-user (4)	Successful User Privilege Gain

Table 7: Rule classification based class type

7.4 Discussion

This paper presents a feasible mechanism to define SLAs guaranteeing the performance of NIDSs. The previous section presented the experimental evaluation showing the validity of the proposed method. In this section, we present some issues that are related to the proposed method.

In ECSLA, the formal language used to describe security monitoring SLAs, tenants specify their security requirements using vulnerabilities for a given application. This level of abstraction is not ideal: it is not easy to know the existing vulnerabilities in an application for regular users. This creates a problem to make the security monitoring SLA life-cycle straightforward. Service providers can cover this issue by offering a separate *vulnerability assessment* service. By using results from such a service, a provider can offer security monitoring SLAs with a higher level of abstraction.

The SLA definition described in this paper uses a fixed set of vulnerabilities, i.e the set of vulnerabilities addressed in a given SLA is fixed. However, in a real situation the number of vulnerabilities that concerns a tenant may increase or decrease. An increase happens when new vulnerabilities concerning a tenant are discovered and a decrease happens when a patch is available for a vulnerability. In our SLA life-cycle both cases are not automated. Notably, the discovery of a new vulnerability could change tenants requirement. Such events are addressed by renegotiating and restarting the process of SLA life-cycle. To facilitate this, it is possible to include termination conditions in the SLA, describing the possibility of renegotiating when new vulnerabilities are discovered.

Moreover, our SLA definition addresses only known vulnerabilities. Monitoring an unknown vulnerability is usually performed by using anomaly-based NIDSs. With a few modifications, our SLA definition could be extended to be used for such a monitoring device. However, as it is in the current state, it cannot be applied to anomaly-based NIDSs because our service description

requires describing the vulnerabilities; this is not practical for currently unknown vulnerabilities. Moreover, guaranteeing the performance of an NIDS covering unknown vulnerabilities is challenging and the risk of not meeting an objective is very high.

While computing the relationship between B and (TPR, FPR), we assumed the resources available for the NIDS are fixed. However, as described in [22], in the cloud environment resources for an NIDS may vary due to elasticity. Our assumption can be interpreted as the minimum amount of resource that is needed to perform the monitoring task. The available resources for an NIDS may increase but will not decrease below the level used for testing. Furthermore, an increase in resources should result in better performance. Hence, an increase in the available resources will not lead to an SLA violation.

The metric estimation method assumes that the provider can conduct a performance test using the lowest guaranteed base rate. In practice, this task is not straightforward. As described in Section 5.1, performing a test using a small base rate value takes a long time and requires large disk space. For example, to measure the performance an NIDS which is configured to monitor the services listed in Table 1 using $B = 10^{-2}$ takes around 42 minutes. Performing the same experiment by changing only the base rate to $B = 10^{-3}$, the experiment takes around four hours. Our previous study [1] presents the actual procedure to perform a test and optimizations to reduce the required time by increasing the degree of parallelism.

To estimate the performance of an NIDS using real attack injection, it requires to have the attack which exploits a given vulnerability. However, getting an attack to exploit a vulnerability is challenging. It is because, usually there is no incentive to publish an attack, especially for commercially owned products. Some issues related to the type of attacks used to perform the test are addressed in [1].

We conclude the paper by presenting a summary in the next section.

8 Conclusion and Future Work

8.1 Conclusion

In this paper, we have discussed the problem of defining a security monitoring SLA. Specifically, we studied SLAs describing the performance of security monitoring probes. To show the proposed SLA definition method, the network security monitoring device we used is a signature-based NIDS.

We started by stating the objective, which is to provide a mechanism that enables the definition of security monitoring SLAs. We continued by listing the problems which need to be addressed in order to meet the objectives and to realize the security monitoring SLAs. The lack of a formal language to define SLAs, finding a relevant KPI i.e. a single unified metric which takes the base rate into account, and reconciling the fact that there are lots of vulnerabilities and the need for a custom-tailored SLAs are presented as problems.

In order to address the lack of a formal language in the SLA definition process, we proposed an extension to CSLA [2]. CSLA is a domain specific language specially designed to describe cloud SLAs. Our extension, called ECSLA, adds the ability to define a security monitoring service which contains the users' requirements description as a list of vulnerabilities. The extension also enables to define a complex parameter which is computed from basic metrics.

To describe the performance of an NIDS we used the C_{ID} as a parameter. C_{ID} meets both of our requirements: it is a single metrics and it takes the base rate into account. Taking the base rate into consideration in SLA comes with two challenges, (i) at what base rate value should a provider offer an SLA as the C_{ID} varies depending on the input base rate ? (ii) in practice measuring the value of the base rate before the occurrence of an attack is very difficult.

To address both of these issues, our SLO definition uses a *model* which takes the base rate as an input. Defining a model addresses both problems: first, it removes the need to use a specific base rate value in the SLA definition; second, the model can estimate the performance metrics for previously unknown base rate values. The model takes a value of B as an input and outputs the *TPR* and *FPR* values. The model is generated by testing the NIDS using known base rate values. Using the results from such tests, and an interpolation-based method, we generate the expected model. The final model can be used as the SLO description in the SLA.

The other issue addressed in this paper is the effect of monitoring multiple vulnerabilities with a single monitoring device. Increasing the number of vulnerabilities in an NIDS results in lower performance. A quantitative measure of the effect between vulnerabilities helps to estimate the performance of the NIDS while aggregating those vulnerabilities together. We introduced the *interference vector and matrix* to describe the effect of vulnerabilities between each other. Using the interference values and the performance for individual vulnerabilities, we presented how to calculate the aggregated metrics. These metrics describe the performance of an NIDS while configured with all the vulnerabilities together.

Having a formal language and the performance estimation method for any group of vulnerabilities can be enough to prepare SLA templates. However, having thousands of vulnerabilities makes it impractical to build the interference matrix. Hence it hinders a custom-tailored SLA template preparation. To address this issue, we proposed a clustering mechanism which groups the vulnerabilities based on some heuristics and performs the interference test per group. The performance of an NIDS on the group indicates the worst performance; hence it gives a lower bound for any subgroup of vulnerabilities from that group. By this approach, it is possible to reduce the dimensions of the interference matrix and to make it practical.

Finally, we showed an experimental evaluation on how to prepare the model that can be used in the security monitoring SLOs. In our experiment, we showed the feasibility of the metrics estimation process using an interpolation-based method. We also showed an example of the clustering method using the Snorts Registered rule set. The providers can perform clustering depending on the available resources. A tradeoff should be maintained between the size of the interference matrix and the number of vulnerabilities per group. Indeed, the performance of an NIDS on a group reduces as the number of vulnerabilities increases.

8.2 Future Work

Some of the issues described in Section 7.4 can be addressed as a future work. The issues of providing a higher level language to describe users need, extending the SLA definition mechanism for monitoring devices other than NIDS and experimental evaluation for the concept of rules interference can be addressed in the future work.

In our SLA definition tenants are expected to describe their needs in terms of software vulnerabilities. This is not an optimal solution because usually, tenants have no detailed information about the vulnerabilities of their software. In order to address this issue, an extension can be done for our SLA definition mechanism. The extension connects our SLA definition mechanism with the vulnerability scanning and fuzzing methods (e.g. OpenVAS [23]). Ideally, users specify the software they need to deploy, the vulnerability scanner and the fuzzer check if there are vulnerabilities or security loopholes in the software and list the existing vulnerabilities with detailed information. Then tenants can select their desired vulnerabilities, and the result can be passed to the SLA definition method.

Practically there are challenges to achieve the connection between vulnerability scanners and SLA definition methods. Specially, considering the diversity of existing applications standardizing the scanning method and communicating with SLA components are few examples.

In addition, generalized automation of a vulnerability scanning method for custom developed applications also requires more study.

In our work we considered a specific security monitoring probe, a signature-based NIDS and Snort [20] is used in our experiments. The output of an NIDS is processed by the *metrics evaluator* component which is not specific to Snort. Any type of NIDS can be used by implementing the appropriate parser to the output of the NIDS. Anomaly-based IDSs could be addressed by extending the SLA definition method. Specifically, the parameters used to describe performance should be adapted for the type of IDS used.

Extending for anomaly-based IDS opens the opportunity to consider monitoring for unknown vulnerabilities. Including unknown vulnerabilities requires extending the service definition section of ECSLA. Currently, a list of known vulnerabilities is defined and the performance of the NIDS is guaranteed on those vulnerabilities.

ECSLA can also be extended to describe the performance of security monitoring probes other than NIDSs (e.g. firewalls). Depending on the device described in the SLA, adaptation may be required for different sections in the SLA, like an adaptation for the description of users requirements, for the guarantees, metrics, and verification method to that specific device. For example, the security monitoring service definition for the firewalls may be expressed in different forms than for the NIDSs and outputs of the firewalls are not the same as the output of NIDS. As a result, a parser implementation can be required to process the output of firewalls.

In our work we introduced the theoretical concept of rules interference (*interference vector and matrix*). A practical evaluation of rule interference is not presented. Collecting a large and enough number of attacks with their corresponding rules to perform experiments would validate our theoretical work. The experiment can be done following the attack injection method presented in [1] and the result can be used to build the interference vector and matrix. In addition, while introducing interference the assumption was an NIDS giving output for every matching rules (as described in Section 6.2.1). Other cases of interference (e.g. NIDS giving outputs for few of the matchings based on some heuristics) are not considered. Having a method with all possible cases would make the formula more consistent and this can be achieved with more studies.

9 Appendix

9.1 XML schema of ECSLA

```
<?xml version ="1.0" encoding ="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
numberstargetNamespace ="http://www.inria.fr/cslamodel "
numberselementFormDefault ="qualified ">

<xs:annotation>
  <xs:appinfo>ECSLA, a DSL for security monitoring SLA description in cloud</xs:appinfo>
  <xs:documentation xml:lang="en">
  Schema for the ECSLA, an extention for CSLA Language.
  Version: 1.0
  Author: Amir Teshome Wonjiga
  </xs:documentation>
</xs:annotation>

<xs:complexType numbersname=" CloudServiceType ">
  <xs:choice >
    <xs:element numbersname=" software " numberstype="csla: SoftwareType "
    numbersminOccurs ="1" numbersmaxOccurs ="1"/>
    <xs:element numbersname=" platform " numberstype="csla: PlatformType "
    numbersminOccurs ="1" numbersmaxOccurs ="1"/>
    <xs:element numbersname=" infrastructure " numberstype="csla: InfrastructureType "
    numbersminOccurs ="1" numbersmaxOccurs ="1"/>
    <xs:element numbersname="macro" numberstype="csla:any" numbersminOccurs ="0"
    numbersmaxOccurs ="1"/>
  </xs:choice >
</xs:complexType >

<xs:complexType numbersname=" SecurityMonitoringType ">
  <xs:sequence >
    <xs:element numbersname=" product " numberstype="csla: ProductType " numbersminOccurs ="1"
    numbersmaxOccurs =" unbounded "/>
    <xs:element numbersname=" vulnerabilities " numberstype="csla: VulnerabilitiesType "
    numbersminOccurs ="0" numbersmaxOccurs =" unbounded "/>
    <xs:element numbersname=" infrastructure " numberstype="csla: InfrastructureType "
    numbersminOccurs ="1" numbersmaxOccurs =" unbounded "/>
  </xs:sequence >
</xs:complexType >

<xs:complexType numbersname=" ProductType ">
  <xs:sequence >
    <xs:element numbersname="mode" numberstype="csla: ModeType " numbersminOccurs ="1"
    numbersmaxOccurs =" unbounded "/>
  </xs:sequence >
  <xs:attribute numbersname="id" numberstype="xs:string "/>
  <xs:attribute numbersname="name" numberstype="xs:string "/>
  <xs:attribute numbersname="mode" numberstype="xs:integer "/>
  <xs:attribute numbersname=" version " numberstype="xs:string "/>
  <xs:attribute numbersname=" distribution " numberstype="xs:string "/>
  <xs:attribute numbersname="price" numberstype="xs:string "/>
  <xs:attribute numbersname=" license " numberstype="xs:string "/>
</xs:complexType >

<xs:complexType numbersname=" VulnerabilitiesType ">
  <xs:sequence >
    <xs:element numbersname=" vulnerability " numberstype="csla: VulnerabilityType "
    numbersminOccurs ="1" numbersmaxOccurs =" unbounded "/>
    </xs:sequence >
```

```
</xs:complexType >

<xs:complexType numbersname=" VulnerabilityType ">
  <xs:attribute numbersname="id" numberstype="xs:string "/>
  <xs:attribute numbersname="cve" numberstype="xs:string "/>
  <xs:attribute numbersname=" description " numberstype="xs:string "/>
</xs:complexType >


<xs:complexType name="MetricType">
  <xs:sequence>
    <xs:element name="description" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="formulation" type="xs:string" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string"/>
  <xs:attribute name="name" type="xs:string"/>
  <!-- Type can be simple of complex -->
  <xs:attribute name="type" type="xs:string"/>
  <xs:attribute name="unit" type="xs:string"/>
</xs:complexType>
</xs:schema >
```

Listing 4: ECSLA XML schema

References

[1] A. Teshome, L. Rilling, and C. Morin, "Verification for security monitoring slas in iaas clouds: The example of a network ids," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–7.

[2] Y. Kouki and T. Ledoux, "Csla: a language for improving cloud sla management," in *International Conference on Cloud Computing and Services Science, CLOSER 2012*, 2012, pp. 586–591.

[3] R. Bejtlich, *The Tao of network security monitoring: beyond intrusion detection*. Pearson Education, 2004.

[4] "Practical guide to cloud service level agreements version 2.0," Cloud Standards Customer Council (CSCC), Whitepaper, 2015.

[5] S. Axelsson, "The base-rate fallacy and its implications for the difficulty of intrusion detection," in *Proceedings of the 6th ACM Conference on Computer and Communications Security*. ACM, 1999, pp. 1–7.

[6] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "Web service level agreement (wsla) language specification."

[7] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Open grid forum*, vol. 128, no. 1, 2007, p. 216.

[8] D. D. Lamanna, J. Skene, and W. Emmerich, "Slang: A language for defining service level agreements," in *NINTH IEEE WORKSHOP ON FUTURE TRENDS OF DISTRIBUTED COMPUTING SYSTEMS, PROCEEDINGS*. IEEE COMPUTER SOC, 2003, pp. 100–106.

[9] R. B. Uriarte, F. Tiezzi, and R. D. Nicola, "Slac: A formal service-level-agreement language for cloud computing," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2014, pp. 419–426.

[10] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing risk and reward in a market-based task service," in *null*. IEEE, 2004, pp. 160–169.

[11] "GFD.224 – Open Cloud Computing Interface – Infrastructure," accessed July 2018. [Online]. Available: http://occi-wg.org/about/specification/

[12] Y. Kouki, "SLA-driven cloud elasticity anagement approach," Theses, Ecole des Mines de Nantes, Dec. 2013. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00919900

[13] P. Wieder, J. M. Butler, W. Theilmann, and R. Yahyapour, *Service level agreements for cloud computing*. Springer Science & Business Media, 2011.

[14] "Common Vulnerabilities and Exposures," accessed July 2018. [Online]. Available: https://cve.mitre.org/

[15] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić, "Measuring intrusion detection capability: an information-theoretic approach," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 90–101.

[16] "Amazon Compute Service Level Agreement," accessed August 2018. [Online]. Available: https://aws.amazon.com/compute/sla/

[17] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[18] "Open source software for creating private and public clouds." accessed July 2018. [Online]. Available: https://www.openstack.org/

[19] "Open vSwitch is a production quality, multilayer virtual switch," accessed July 2018. [Online]. Available: https://www.openvswitch.org/

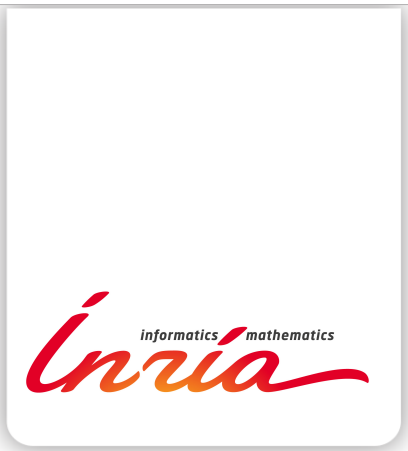
[20] "An open source intrusion detection and prevention system," accessed August 2018. [Online]. Available: https://www.snort.org/

[21] E. Raftopoulos and X. Dimitropoulos, "A quality metric for ids signatures: in the wild the size matters," *EURASIP Journal on Information Security*, vol. 2013, no. 1, p. 7, 2013.

[22] A. Milenkoski, K. Jayaram, N. Antunes, M. Vieira, and S. Kounev, "Quantifying the attack detection accuracy of intrusion detection systems in virtualized environments," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on.* IEEE, 2016, pp. 276–286.

[23] "Open Source vulnerability scanner and manager (OpenVAS)," accessed July 2018. [Online]. Available: http://www.openvas.org/

Contents



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399